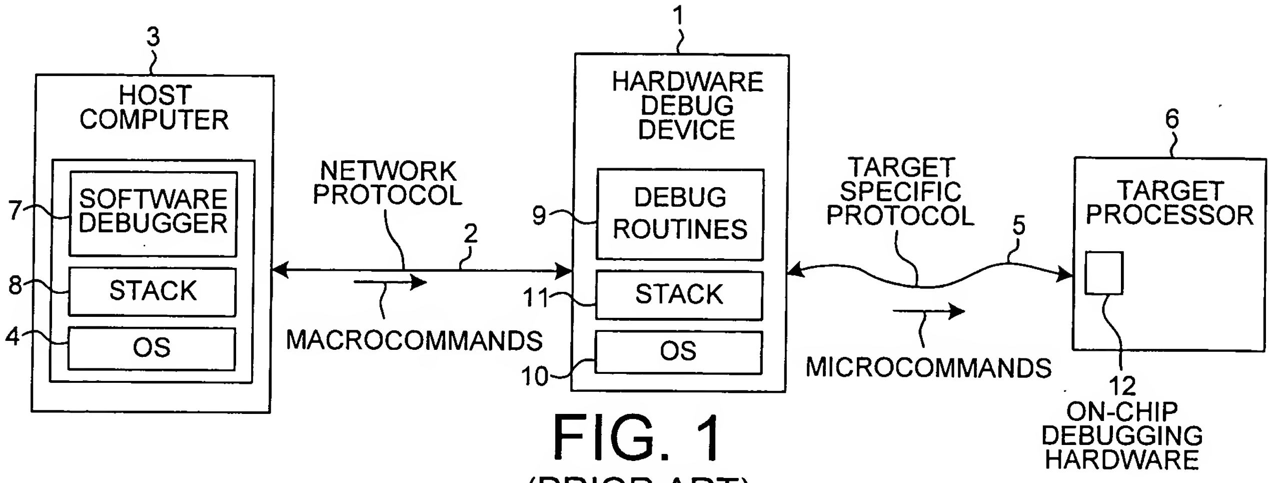

FIG. 1
(PRIOR ART)

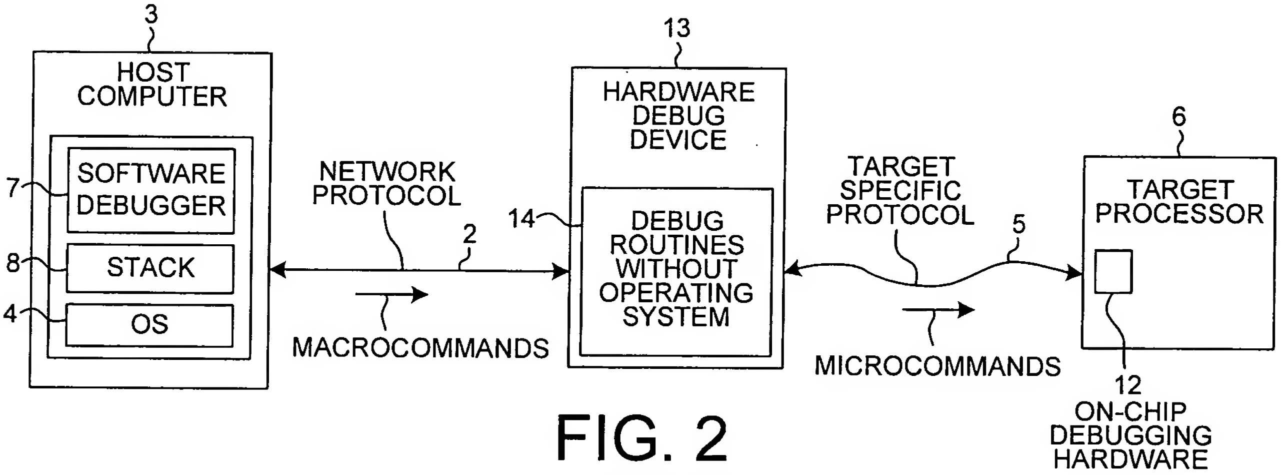

FIG. 2
(PRIOR ART)

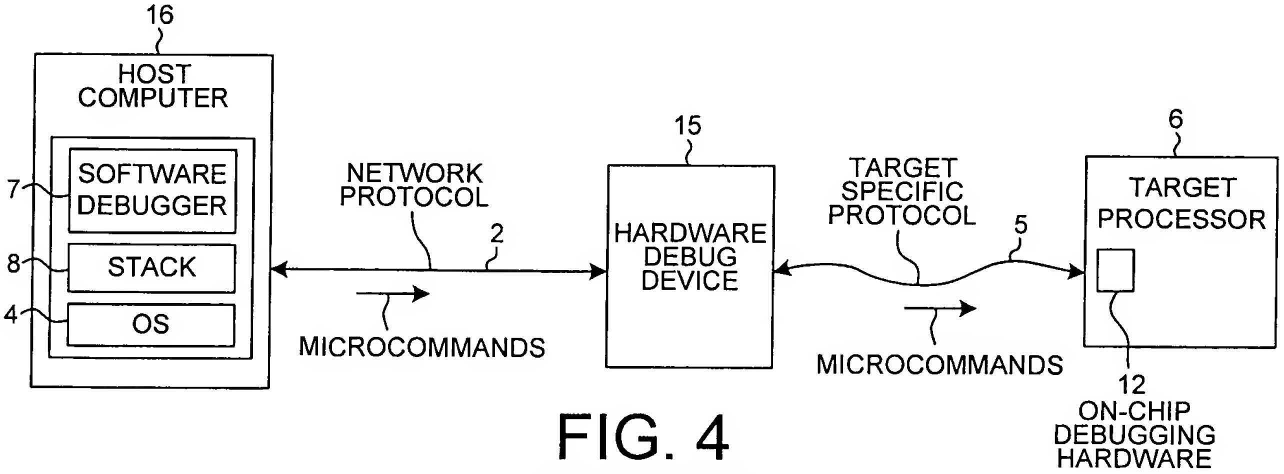

FIG. 4
(PRIOR ART)

```
HOST COMMAND:  CC_SETMEMORY(Address, len, data)
/**
*  Writes data to the target's physical memory based on the provided data (parameter px).
PXFERSPDU is a structure used to hold the address, data and other specifics needed to carry out the
write.
*/
void msWritePhysical (PXFERSPDU px)
{
    BOOL fCancelled;
    short wResplen;
    ULONG lAddress;
    USHORT nBytes, wDatalen;
    UCHAR *pSrc;

    /* initialize */
fCanceled = FALSE;
wResplen = 0;

/* set data length from px param */
wDatalen =
    px->pktlen -
    sizeof(px->spdu.memsetl.address) -
    sizeof(px->spdu.memsetl.array.nbytes);
/* set address to write to from px param */
lAddress = ulswap(px->spdu.memsetl.address);
/* set number of bytes to write from px param */
nBytes = unswap(px->spdu.memsetl.array.nbytes) + 1;
/* set data to write from px param */
pSrc = px->spdu.memset1.array.bytes;

/* save the values of the PC and ADL */
SaveReg((ULONG)(EZ80_PC_MASK | EZ80_ADL_MASK));
/* force ADL mode */
tpSetRegisterByte(EZ80_ADL_MASK, 1);
/* write the data to target memory */
z8WritePhy(lAddress, nBytes, pSrc, wDatalen);
/* restore PC and ADL */
RestoreReg(ULONG)(EZ80_PC_MASK | EZ80_ADL_MASK));

/* return the response of the request was not cancelled */
if (!fCancelled)
    xioRespond(wResplen);
}

/**
* Write to physical memory.
*/
BOOL Z8WritePhy(ULONG lAddress, USHORT, nBytes, UCHAR *pBuff, USHORT wDatalen)
{
    BOOL fCancelled = FALSE;
    UCHAR *pSrc;
    USHORT nDatacnt, nHostBrkCnt;

    nDataCnt = 0;
    nHostBrkCnt = HOST_BRK_CNT;
    pSrc = pBuff;
```

FIG. 3A
(PRIOR ART)

```
    /* Set starting adr... */
    tpSetRegisterLong(EZ80_PC_MASK, lAddress);

    /* for the number of bytes to be written, write each byte */
    do {
        /* write a byte to target */
        tpWriteMemAtPC(*pSrc++);

        /** Loop through source buffer until requested number of bytes written.  */
        nDatacnt++;
        if (nDatacnt >= wDatalen)
        {
            nDatacnt = 0;
            pSrc = pBuff;
        }

        if (--nHostBrkCnt == 0)
        {
            nHostBrkCnt = HOST_BRK_CNT;
            if (fCancelled = xioHostBreak(0))
                break;
        }
    } while (--nBytes);

    return(fCancelled);
}

/** Write a byte at the address held by the PC **/
void tpWriteMemAtPC(BYTE bData)
{
    ZDIWrite(ZDIW_MEMORY, bData);
}

/** Write the given byte to the target at the given address **/
void ZDIWrite(UCHAR address, UCHAR data)
{
    TURN_LED_ON(LED_COMM);
    /* this does the actual write of the byte */
    *pZdiDataPort = data;
    ckZDIbusy();
    WriteZDIaddr(address);
    TURN_LED_OFF(LED_COMM);
}

/** Write an address to the target **/
void WriteZDIaddr(UCHAR addr)
{
    /* this does the actual setting of the address **/
    *pZdiAdrPort = addr;
    ckZDIbusy();
}
```

KEY TO FIG. 3

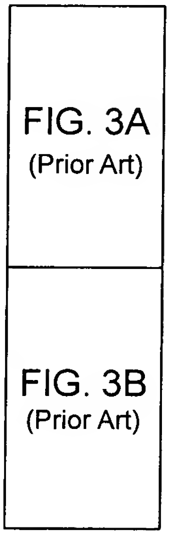

FIG. 3B
(PRIOR ART)

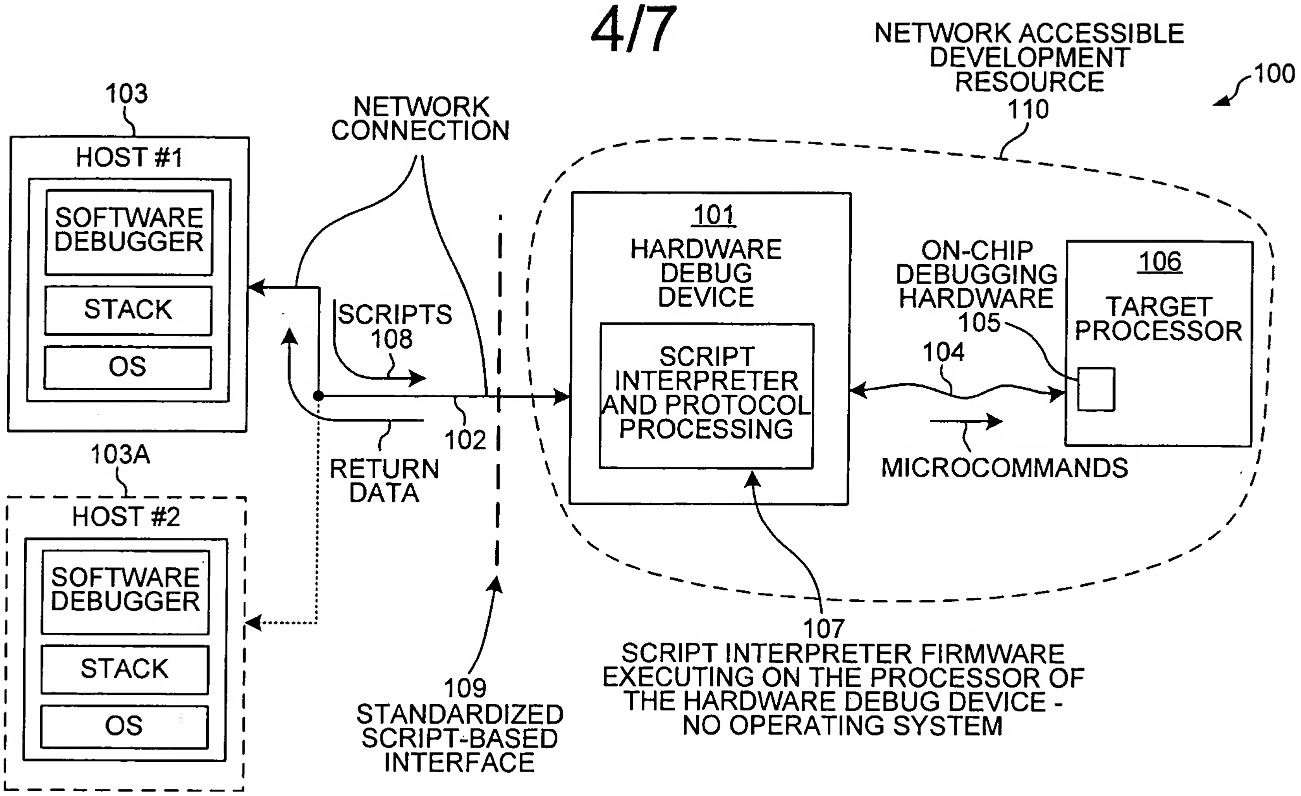
NETWORK ACCESSIBLE DEVELOPMENT RESOURCE
110
100
103
HOST #1
SOFTWARE DEBUGGER
STACK
OS
NETWORK CONNECTION
SCRIPTS
108
102
RETURN DATA
101
HARDWARE DEBUG DEVICE
SCRIPT INTERPRETER AND PROTOCOL PROCESSING
ON-CHIP DEBUGGING HARDWARE
105
104
106
TARGET PROCESSOR
MICROCOMMANDS
103A
HOST #2
SOFTWARE DEBUGGER
STACK
OS
109
STANDARDIZED SCRIPT-BASED INTERFACE
107
SCRIPT INTERPRETER FIRMWARE EXECUTING ON THE PROCESSOR OF THE HARDWARE DEBUG DEVICE - NO OPERATING SYSTEM

FIG. 5

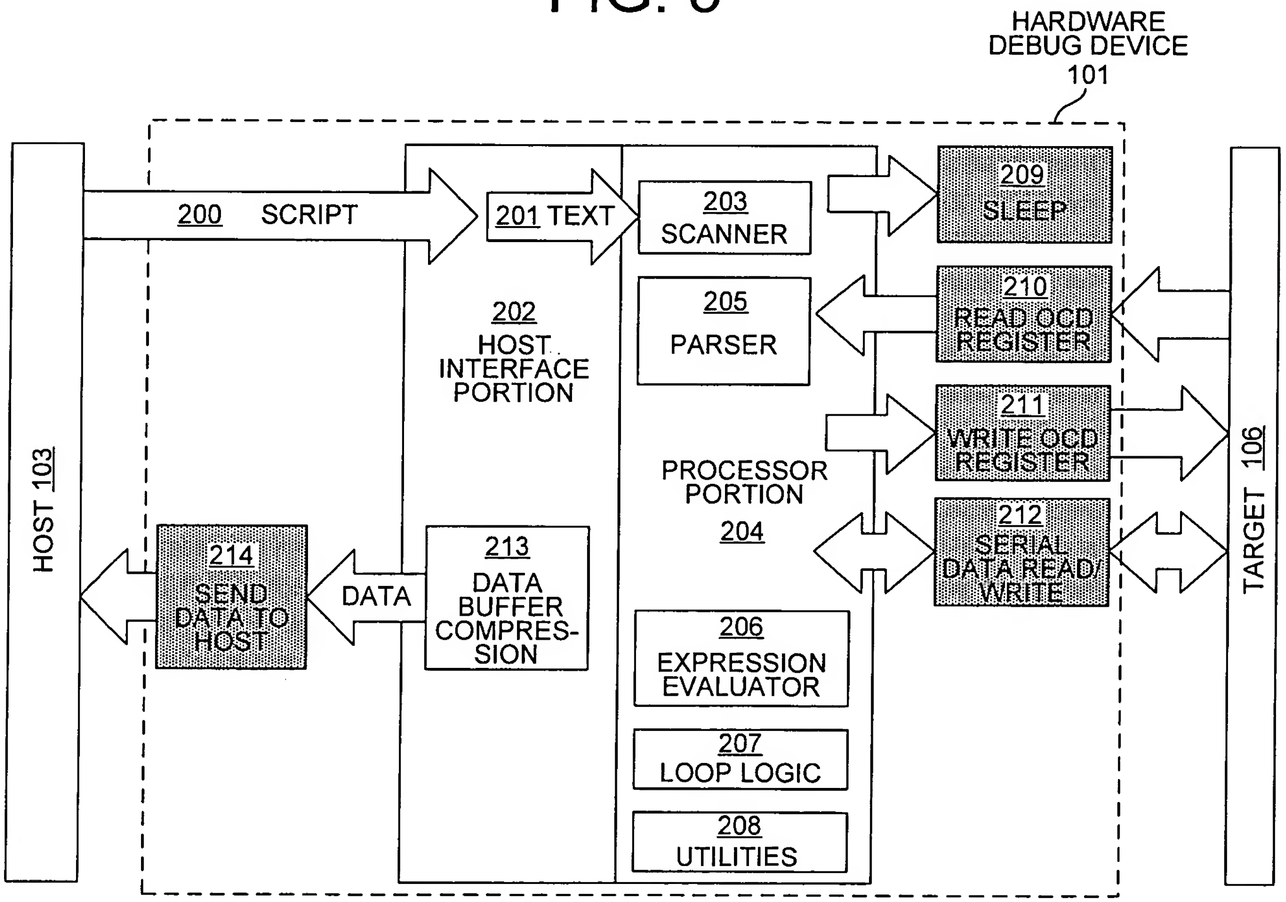
HARDWARE DEBUG DEVICE
101
HOST 103
200
SCRIPT
201 TEXT
202
HOST INTERFACE PORTION
203
SCANNER
205
PARSER
PROCESSOR PORTION
204
206
EXPRESSION EVALUATOR
207
LOOP LOGIC
208
UTILITIES
209
SLEEP
210
READ OCD REGISTER
211
WRITE OCD REGISTER
212
SERIAL DATA READ/WRITE
213
DATA BUFFER COMPRES-SION
DATA
214
SEND DATA TO HOST
TARGET 106

FIG. 6

```
R16=07V07=R10|(R11<<8)|(R12<<10)                ;  Save PC in DTLi variable V07
  V06=(R03&10)>>4                               ;  Save ADL in DTLi variable V06
R15=UAddr R14=HiAddr R13=LoAddr R16=87          ;  Set start memory location
R16=(Adl&01)?08:09                              ;  Set current ADL
.[]=data_byte_count:data                        ;  Set the data to the internal buffer
R30=[0:@]                                       ;  Write Buffer to target memory
R13=V07R14=V07>>8R15=V07>>10                    ;  Resore PC
R16=87R16=(V06&1)?08:0                          ;  Restore ADL
```

FIG. 7

```
R16=07V07=R10|(R11<<8)|(R12<<10)V06=(R03&10)>>4R15=0R14=0R13=0R16=87R16=(1&01)?
08:09RFA=01.[]=9:7f7f7f7f7f7f7f7f7fR30=[0:@]
RFA=00R13=V07R14=V07>>8R15=V07>>10R16=87R16=(V06&1)?08:0
```

FIG. 8

PRECEDENCE	OPERATORS	ASSOCIATIVITY
1	[ ]	LEFT
2	! ~ ++ - - - (UNARY)	RIGHT
3	* / %	LEFT
4	+ -	LEFT
5	<< >>	LEFT
6	< <= > >=	LEFT
7	== !=	LEFT
8	&	LEFT
9	^	LEFT
10	|	LEFT
11	&&	LEFT
12	||	LEFT
13	?:	RIGHT
14	=	RIGHT

FIG. 9

BUFFER+0	BUFFER+1	BUFFER+2	MEANING
'\x01' .. '\x0f'			Repeat ASCII hexadecimal byte (buffer+1 & buffer+2) the number of times specified by buffer+0. Up to 15 times.
'\x81'	nn		Repeat ASCII hexadecimal byte (buffer+2 & buffer+3) the number of times specified by buffer+1 (binary nn). Up to 256 times.
'\x82'	nn	mm	Repeat ASCII hexadecimal byte (buffer+3 & buffer+4) the number of times specified by (buffer+1 <<8)|(buffer+2)(binary nnmm). Up to 65535 times.
ANY OTHER CHARACTER			The data byte is the ASCII hexadecimal value of buffer+0 and buffer+1

FIG. 10

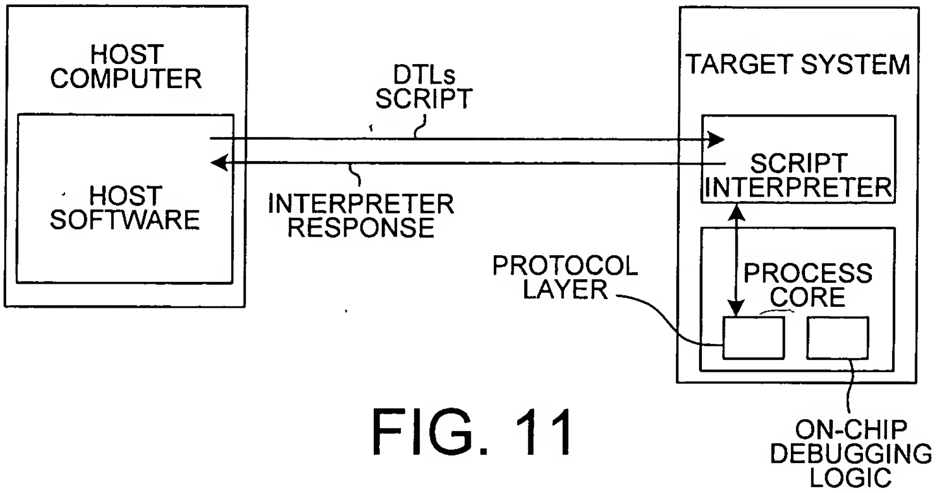
HOST COMPUTER
HOST SOFTWARE
DTLs SCRIPT
INTERPRETER RESPONSE
TARGET SYSTEM
SCRIPT INTERPRETER
PROTOCOL LAYER
PROCESS CORE
ON-CHIP DEBUGGING LOGIC

FIG. 11

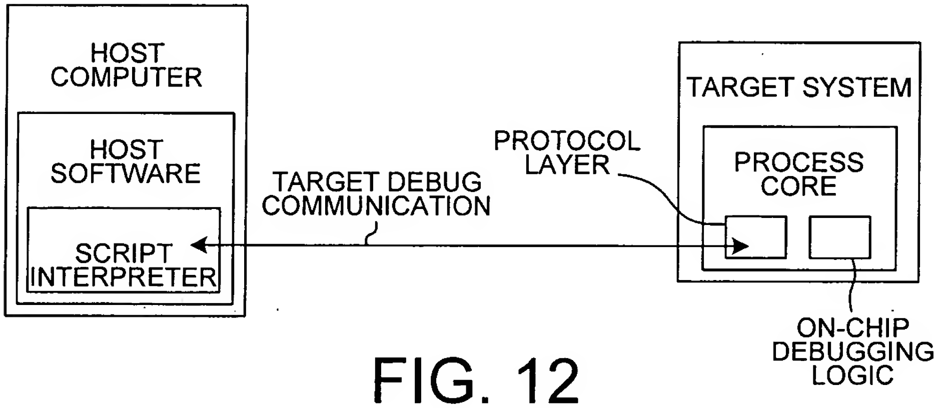
HOST COMPUTER
HOST SOFTWARE
SCRIPT INTERPRETER
TARGET DEBUG COMMUNICATION
PROTOCOL LAYER
TARGET SYSTEM
PROCESS CORE
ON-CHIP DEBUGGING LOGIC

FIG. 12